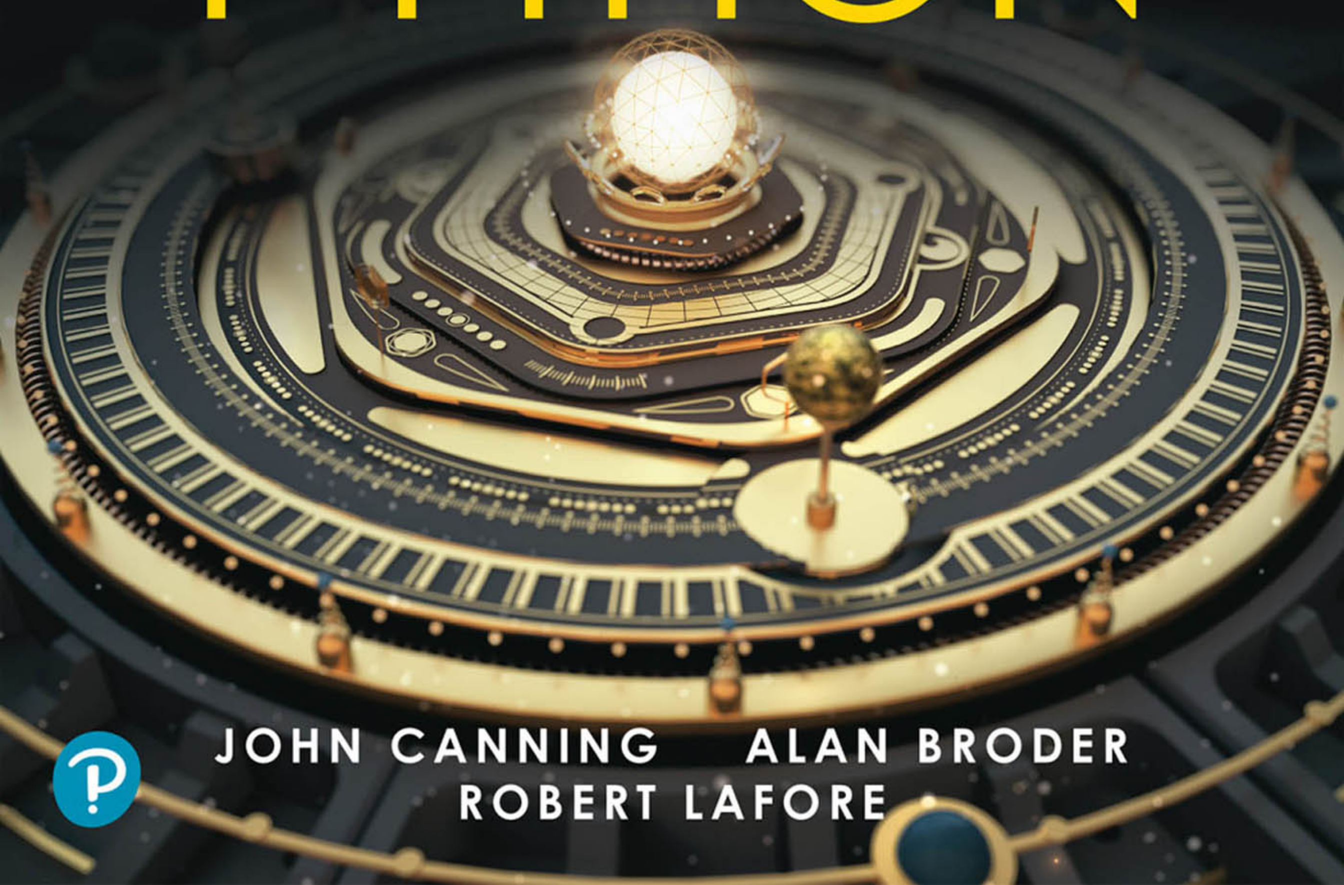
DATA STRUCTURES & ALGORITHMS in PYTHON



JOHN CANNING ALAN BRODER
ROBERT LAFORE

John Canning
Alan Broder
Robert Lafore

Data Structures & Algorithms in Python

Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

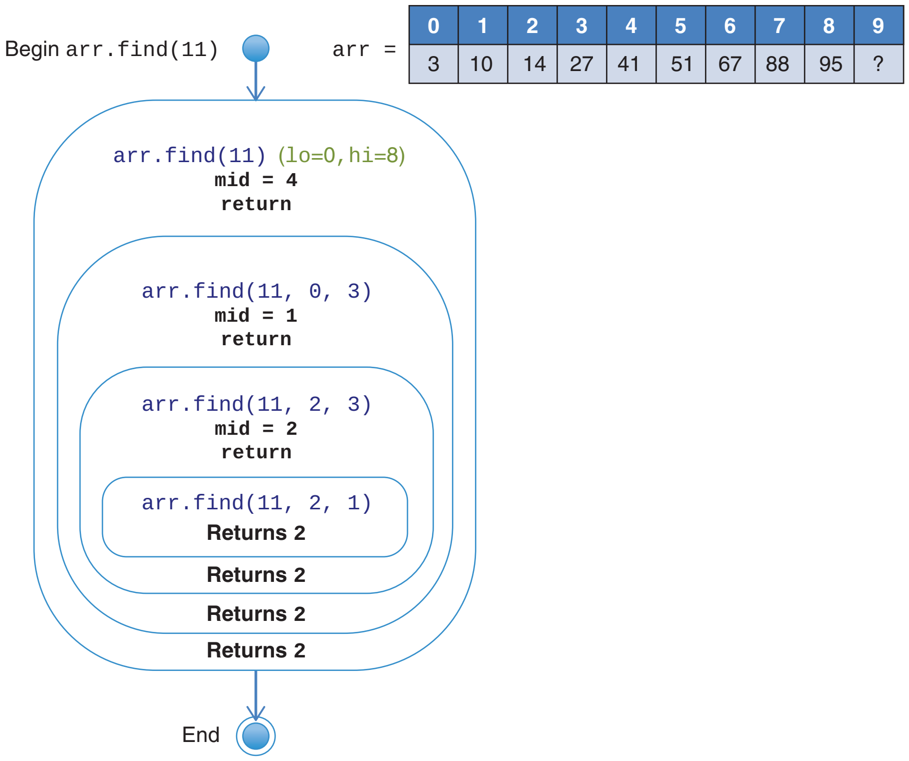


FIGURE 6-8 Performing recursive binary search

Divide-and-Conquer Algorithms

The recursive binary search is an example of the *divide-and-conquer* approach. You divide the big problem into smaller problems and solve each one separately. The solution to each smaller problem is the same: you divide it into even smaller problems and solve them. The process continues until you get to the base case, which can be solved easily, with no further division.

The divide-and-conquer approach is commonly used with recursion, although, as you saw in the binary search in Chapter 2, you can also use a nonrecursive approach. It usually involves a method that contains two recursive calls to itself, one for each half of the problem. In the binary search, there are two such calls, but only one of them is actually executed. (Which one depends on the value of the keys.) The mergesort, which is described later in this chapter, actually executes both recursive calls (to sort two halves of an array).

The Tower of Hanoi

The Tower of Hanoi is a historic puzzle consisting of several disks placed on three spindles or columns, as shown in Figure 6-9. The puzzle is also known as the Tower of Brahma or Lucas's Tower for Édouard Lucas, its inventor. Lucas was a mathematician who studied the

Fibonacci sequence and other recursively defined sequences. The goal of the puzzle is to move all of the disks from one spindle to another, following a set of specific rules.

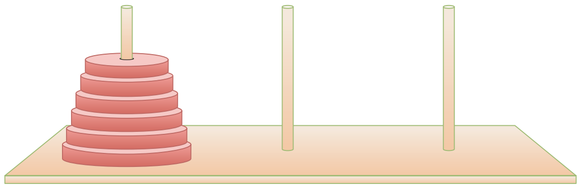


FIGURE 6-9 The Tower of Hanoi puzzle

The disks all have different diameters. Each one has a hole in the middle that fits over the spindles. All the disks start out on one spindle, stacked in order of diameter, which gives the appearance of a tower. The object of the puzzle is to transfer all the disks from the starting spindle, say the one on the left, to another spindle, say the one on the right. Only one disk can be moved at a time, and no disk may be placed on a disk that's smaller than itself.

The legend that goes along with the puzzle is that in a distant temple, a group of monks labor day and night to transfer 64 golden disks from one of three diamond-studded towers to another. When they are finished, the world will end. If that alarms you, wait until you see how long it takes to solve the puzzle with 64 disks.

The TowerofHanoi Visualization Tool

Start up the TowerOfHanoi Visualization tool. Create a new three-disk puzzle by typing 3 in the text entry box and selecting New. You can attempt to solve the puzzle yourself by using the mouse to drag the topmost disk to another tower. The star by the spindle on the right indicates the goal tower. Figure 6-10 shows how the towers look after several moves have been made.

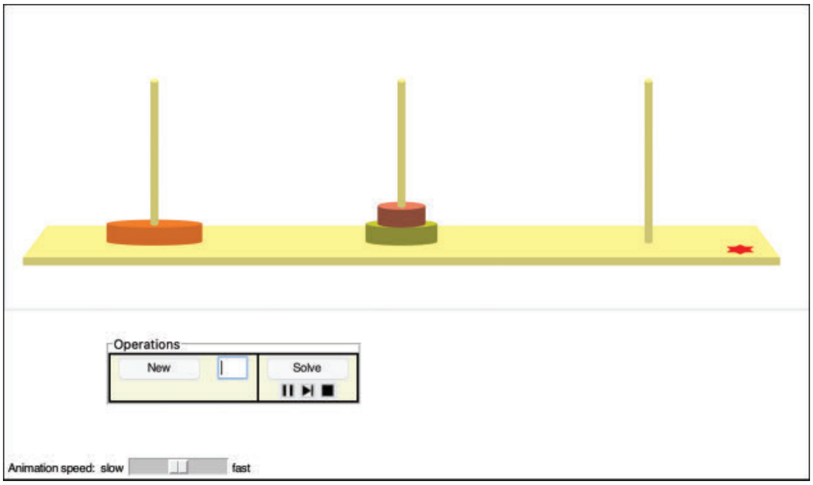


FIGURE 6-10 The TowerOfHanoi Visualization tool

There are two ways to use the visualization tool:

- You can attempt to solve the puzzle manually, by dragging the disks from tower to tower.
- You can select the Solve button and watch the algorithm solve the puzzle with no intervention on your part. The disks zip back and forth between the posts. Using the Pause/Play button, you can stop and resume the animation to examine what happens at each step.

To restart the puzzle, type in the number of disks you want to use, from 1 to 6, and select New. The specified number of disks will be arranged on the left spindle. You can drag the top disk to either of the other two spindles. If you pick up the next larger disk, the tool will allow you to place it only on a spindle whose topmost disk has a larger diameter. If you release it away from a spindle that can accept it, the disk returns to where you picked it up. When all the disks form a tower on the leftmost spindle, the Solve button is enabled. If you stop the automated solver, you can resume manually solving the puzzle.

Try solving the puzzle manually with a small number of disks, say three or four. Work up to higher numbers. The tool gives you the opportunity to learn intuitively how the problem is solved.

Moving Pyramids

Let's call the arrangement of disks on a spindle a "pyramid" because they are ordered by diameter. You could call them a "stack," but that would be somewhat confusing with the stack data structure you've studied (although that data structure will be useful to model the disks). Using this terminology, the goal of the puzzle is to move the pyramid from the left spindle to the right. For convenience, the spindles can be labeled L, M, and R for left, middle, and right.

How do you solve the puzzle? Let's start with the easy cases. If there's only one disk, the solution is trivial. Move the one disk from spindle L to spindle R. In fact, even easier, if there are no disks, then there is nothing to move and the puzzle is solved. Then what about two disks? That's pretty easy, too. Move disk 1 from L to M, move disk 2 from L to R, and then move disk 1 from M to R. These moves are shown in the four panels of Figure 6-11.

How do you solve problems with more disks? You've seen the base cases; can recursion help with this? You might be surprised to know that you've already figured out all the steps of the recursive algorithm by enumerating the cases up to two disks.

To see how to get to a recursive solution, remember that you need to make smaller versions of the same problem. The two strategies you've seen are dividing the problem in half or reducing the number of things by one. Dividing it in half doesn't seem to fit, so let's look at reducing the number of disks by one. In particular, let's look at the case of three disks.

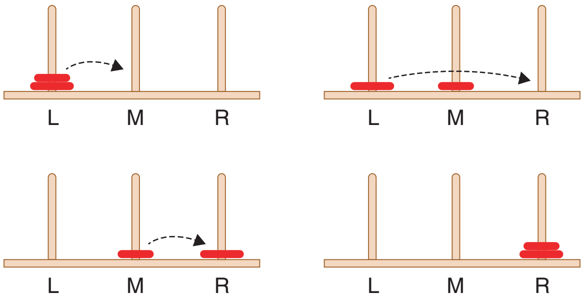


FIGURE 6-11 Solution to the Tower of Hanoi with two disks

If you perform the same steps that you took to solve the two-disk puzzle, then the three-disk puzzle would end up with disk 3 still on the left spindle and disks 1 and 2 on the right spindle, as shown in Figure 6-12. From here, it's easy to move disk 3 onto the middle spindle, but not easy to move it to the right-hand one.

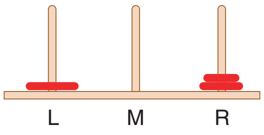


FIGURE 6-12 A Tower of Hanoi with three disks after moving two of them

If you changed the first moves to swap the middle and right spindles, then disks 1 and 2 would end up on the middle spindle and leave the right one open, as shown in Figure 6-13.

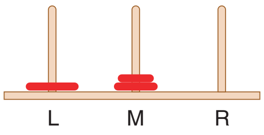


FIGURE 6-13 A Tower of Hanoi after moving two disks to the middle

With the right spindle empty, you can easily move disk 3 from the left to the right. Now all that's left is moving the two disks on the middle spindle over to the right. And that's a problem you've already solved. This example provides the basic outline for the solution of the three-disk problem, namely:

1. Move the top two disks to the middle spindle (using the right one as the spare).
2. Move disk 3 to the right spindle.
3. Move the top two disks on the middle spindle to the right spindle (using the left one as the spare).

Steps 1 and 3 are solutions that have the same form as the three moves shown in Figure 6-11. They are solutions to the two-disk problem, but with different starting and ending positions. Now it's time to recognize that you've solved the three-disk problem by reducing it to two calls to the two-disk solution plus one disk move in between. If you rephrased the preceding outline to solve an *N* disk problem, it would be

1. Solve the *N*–1 disk problem by moving the *N*–1 disk pyramid from the start spindle to the nongoal spindle (using the goal spindle as a spare).
2. Move the *N*th disk to the goal spindle.
3. Solve the *N*–1 disk problem by moving the *N*–1 disk pyramid from step 1 on to the goal spindle (using the starting spindle as a spare).

Steps 1 and 3 look like recursive calls to the same solution routine. That solution routine would need to know how many disks to move and the role of each of the three spindles: start spindle, goal spindle, and spare spindle. In the various recursive calls, the roles of the spindles change. If you think about the difference between Figure 6-12 and Figure 6-13, they perform the same solution but with the swapped roles of the right two spindles.

Is that it? Have you figured out all the steps? What about all the other, bigger disks? Wouldn't they prevent the solution from working because they were in the way of one of the steps?

If you think about it, it doesn't matter that there are more disks in the puzzle because they are all larger than the topmost disks. The outlined solution applies only to the top *N* disks, and disks *N*+1, *N*+2, and so on all must be larger. For example, applying the two-disk solution to move from the left to right spindles works equally well in all the situations shown in Figure 6-14, which has five disks. You count on that fact in the outlined solution. In fact, you hope the situation looks like the rightmost panel of Figure 6-14 at some point during the *N* = 5 puzzle. If all five disks started on the left spindle, performing all the outline steps for *N* = 5 and *N* = 4 and then steps 1 and 2 when *N* = 3 should leave disk 3 on the right (goal) spindle with disks 4 and 5 underneath it. All that remains is performing step 3 on the remaining *N*–1 (two) disks.

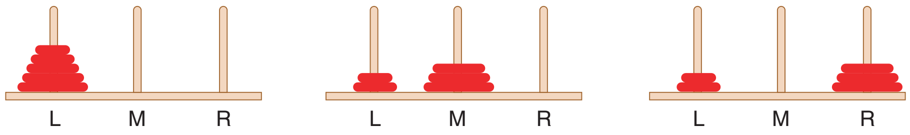


FIGURE 6-14 Possible states in a five-disk Tower of Hanoi

It still seems as though the algorithm might ask to move a larger disk on top of a smaller one after all the swapping of roles of the spindles. How can you tell that it won't try to move a disk from the right spindle to the left in the rightmost condition shown in Figure 6-14? There are two ways: write a program to test it and use mathematical induction to prove it. We tackle that program next.

The Recursive Implementation

Before diving into the code, let's look at what base cases are needed. As we mentioned, if there are no disks, no moves are needed, and the solution is done. That's definitely a base case. Do you need a base case for a single disk? You can check by seeing what happens if you apply the outline recursive solution to the case where *N* is 1. Step 1 is to solve the movement of the *N*–1 disk problem. Because *N*–1 is 0, you know that nothing will be moved. That's followed by moving the *N*th disk to the goal spindle, which means the one disk is moved to the final position. Then step 3 also does nothing (sometimes called a "no-op"), because *N*–1 is 0. Hence, there doesn't seem to be a need to write anything special for the case where *N* is 1. You only need the base case for 0 and the recursive outline.

Listing 6-2 shows the first part of a class, `TowerOfHanoi`, that solves the puzzle for any number of disks. Each instance of the puzzle will have a specific number of disks that is provided to the constructor. The puzzle needs to keep track of what disks are on what spindles. For that, you could use the `SimpleStack` class that was implemented in Chapter 4, "Stacks and Queues." A stack is perfect for modeling each spindle because the only allowed disk movements involve the top of the stack/spindle. You create a stack for each of the three spindles and push integers on it to represent the diameters of the disks. The diameters can be the numbers from 1 up to *N*, the number of disks. The bigger numbers are the larger diameter disks.

LISTING 6-2 The `TowerOfHanoi.py` Module—Puzzle Object

```
from SimpleStack import *

class TowerOfHanoi(object):      # Model the tower on 3 spindles using
                                 # 3 stacks
   def __init__(self, nDisks=3): # Constructor w/ starting number of
      self.__stacks = [None] * 3 # Stacks of disks
      self.__labels = ['L', 'M', 'R'] # Labels for stacks/spindles
      self.__nDisks = nDisks     # Total number of disks
      self.reset()

   def reset(self):              # Initialize state of puzzle
      for spindle in range(3):   # Set up each of 3 spindles
         self.__stacks[spindle] = Stack( # Start w/ empty stack
            self.__nDisks)       # that can hold all the disks
         if spindle == 0:        # On the first spindle,
            for disk in range(   # push the disks on the stack
                  self.__nDisks, 0, -1): # in descending order of size
               self.__stacks[spindle].push(disk)

   def label(self, spindle):     # Get the label of spindle
      return self.__labels[spindle]
```